Scalable mpNoC for massively parallel systems - Design and implementation on FPGA

Mouna Baklouti, Yassine Aydi, Philippe Marquet, Jean-Luc Dekeyser, Mohamed Abid

Scalable mpNoC for Massively Parallel Systems – Design and Implementation on FPGA

M. Baklouti[*,a,b], Y. Aydi[a], Ph. Marquet[b], J.L. Dekeyser[b], M. Abid[a]

[a] *CES Laboratory, National Engineering School of Sfax, Sfax, Tunisia*
[b] *Univ. Lille, F-59044, Villeneuve d'ascq, France*
LIFL, Univ. Lille 1, F-59650, Villeneuve d'ascq, France
INRIA Lille Nord Europe, F-59650, Villeneuve d'ascq, France
UMR 8022, CNRS, F-59650, Villeneuve d'ascq, France

Abstract

The high chip-level integration enables the implementation of large-scale parallel processing architectures with 64 and more processing nodes on a single chip or on an FPGA device. These parallel systems require a cost-effective yet high-performance interconnection scheme to provide the needed communications between processors. The massively parallel Network on Chip (mpNoC) was proposed to address the demand for parallel irregular communications for massively parallel processing System on Chip (mppSoC). Targeting FPGA based design, an efficient mpNoC low level RTL implementation is proposed taking into account design constraints. The proposed network is designed as an FPGA based IP (Intellectual Property) able to be configured in different communication modes. It can communicate between processors and also perform parallel I/O data transfer which is clearly a key issue in an SIMD system. The mpNoC RTL implementation presents good performances in terms of area, throughput and power consumption which are important metrics targeting an on chip implementation. MpNoC is a flexible architecture that is suitable for use in FPGA based parallel systems. This paper introduces the basic mppSoC architecture. It mainly focuses on the mpNoC flexible IP based design and its implementation on FPGA. The integration of mpNoC in mppSoC is also described. Implementation results on a StratixII FPGA device are given for three data parallel applications ran on mppSoC. The obtained good performances justify the effectiveness of the proposed parallel network. It is shown that the mpNoC is a lightweight parallel network making it suitable for both small as well as large FPGA based parallel systems.

Key words: Communication, FPGA, Network architecture, SIMD parallel processing, System-on-a-Chip.

1. Introduction

Modern applications like audio/video compression, image processing and 3D graphics, need high performances and efficient architectures to be executed, especially in the embedded systems domain. These applications require high execution speed and often real time processing capabilities. In the nineteens, designers focus on SIMD (Single Instruction Stream Multiple Data Stream) architectures. This allows them to speed-up significantly execution times of a variety of tasks from matrix multiplication to image processing. Typically an SIMD machine has a control unit which broadcasts instructions to N processors, numbered (addressed) from 0 to N-1, and all active processors execute the same instruction at the same time [32]. The interconnection network (ICN) can be used to connect processors to memory modules. Alternatively, the network can be used to interconnect processing elements (PE), where each PE consists of a processor and a memory module. But, until the last few years, silicon integration technology doesn't allow us to put such systems to the SoC context, consequently these systems were then less and less used. Since recently, we are able to integrate billions transistors in a single chip (giga and tera-scale integration GSI/TSI). On one side, new design methodologies such as IP (Intellectual Property) reuse and, on the other side, the possible high integration level on a

[*]Corresponding author
Email addresses: mouna.baklouti@lifl.fr (M. Baklouti), yassine.aydi@oous.rnu.tn (Y. Aydi), philippe.marquet@lifl.fr (Ph. Marquet), jean-luc.dekeyser@lifl.fr (J.L. Dekeyser), mohamed.abid@enis.rnu.tn (M. Abid)

chip let us envisage such a revival of SIMD architectures. Nowadays we have a great variety of high capacity programmable chips, also called reconfigurable devices (FPGAs) where we can easily integrate complete SoCs architectures for many different applications. Due to the inherent flexibility of these devices (Field-Programmable), designers are able to quickly develop and test several hardware/software architectures. In this paper, we used Altera reconfigurable devices to implement the mppSoC (massively parallel processing System on Chip) architecture and get experimental results.

The complexity in circuit design grows rapidly, still validating Moores Law. Therefore, the ability of implementing complex architecture in a single chip always presents new challenges. One of the issues met by designers when implementing large SoCs is the communication between their (numerous) components. Interconnects are considered as one of the most important challenges in GSI/TSI facing today. At the chip level, interconnects have become the major component in the delay of critical paths, are the largest source of power dissipation, and cause reliability problems [33]. So an efficient ICN intra-chip, with low latency and high bandwidth, is fundamental for high communication throughput among one components'architecture. In fact, sharing busses is no longer a good method for connecting multiple processors or other IPs due to their lack of scalability and important power consumption. An alternative to bus based communication is network-on-chip (NoC) [17]. NoC is an emerging paradigm for communications within multiprocessor systems implemented on a single silicon chip.

Our contribution to SIMD on-chip design domain consists in the implementation at an RTL abstraction level of a system called mppSoC [29]. It looks like an on-chip version of the famous MasPar [35]. MppSoC contains a number of processing elements (PE), each one has its own private local data memory. The whole system is mastered by a processor called Array Controller Unit (ACU). The PEs communicate using a Xnet neighborhood ICN. It permits regular and neighboring communications between PEs, but is not really efficient when communications become irregular. Thus, to improve the communication performances of the mppSoC, an additional communication network, to assure point to point communications, is needed. The objective of this work is to provide designers with more complete parallel architecture with high performances. Our contribution is to propose a flexible massively parallel NoC, implemented as a VHDL IP, to be integrated in such particular systems. The mpNoC can be configured to support different communication modes by programming. The communication instructions permitting to manage the global router, the PEs and the ACU are defined. The present paper concentrates on point to point communications. Two types of global router, crossbar and delta multistage networks (MIN), are discussed in detail.

The rest of this paper is organized as follows. The use of a NoC in parallel architectures state of the art is proposed in the next section. Section 3 introduces the mppSoC platform. The mpNoC implementation is detailed in Section 4. In this Section, we propose an efficient way to integrate the communication network with the mppSoC processors and devices. The execution of some representative applications varying mppSoC communication networks is briefly described in Section 5. Section 6 discusses the performance results of the mpNoC compared to other NoC implementations. Section 7 outlines the conclusion with planned future work.

2. Related Works

The research effort in performance evaluation of communication scheme of multiprocessor SoCs and especially NoC has been widely tackled in order to guarantee optimal communication performance. There has also been a growing interest in designing novel NoC alternatives for parallel architectures. Parallel systems have been designed around a variety of inter processor communication networks. Due to rapid advancement in VLSI technology, it has become feasible to construct massively parallel systems based on static interconnection structures as meshes, trees and hypercubes. Another class of parallel systems includes crossbar, multistage switching networks such as delta network, etc. These mechanisms exhibit various trade-offs between processor throughput, communication delays, and the programming complexity. The performance of ICN under uniform traffic load has been studied with both analytical methods and simulations ([16], [13], [10], [6], [11]).

SIMD implementations usually consist of a Control Unit, an arbitrary number of PE and an ICN, which often are custom-made for the type of application it is intended for. Processors can communicate with each other through the ICN. If the ICN does not provide direct connection between a given pair of processors, then this pair can exchange data via an intermediate processor. The ILLIAC IV [3] used such an interconnection scheme. It is composed of 64 processors operated on 64-bit words. The ICN in the ILLIAC IV allowed each processor to communicate directly with four neighboring processors in an 8x8 matrix pattern such that the i^{th} processor can communicate directly with the four

neighbours: $(i-1)^{th}$, $(i+1)^{th}$, $(i-8)^{th}$ and $(i+8)^{th}$ processors. So, to move data between two PEs, that are not directly connected, the data must be passed through intermediary PEs by executing a programmed sequence of data transfers [2]. This can lead to excessive execution time if more irregular communications are needed. According to [14] the ILLIAC network is considered as a single stage SIMD network. MorphoSys [26] [5] is a reconfigurable SIMD architecture that targets for portable devices. It combines an array of 64 Reconfigurable Cells (RCs) and a central RISC processor (TinyRISC) so that applications with a mix of sequential tasks and coarse-grain parallelism, requiring computation intensive work and high throughput can be efficiently implemented on it. In MorphoSys, each RC can communicate directly with its upper, below, left and right neighbors peer to peer. This gives efficient regular applications, but unfortunately non-neighbors communications seem to be tedious and time consuming. Other new SIMD architectures have been proposed [24] [34] but they don't solve the problem of irregular communications since they integrate only a neighborhood ICN. Among popular interconnected networks there are tree and hypercube structures. Tree type structure is suited to certain special kind of parallel processing where neighbouring processors can communicate quite fast. However, communication between non neighbouring processors is slower and it requires intermediate processors to store and forward message transfer. The hypercube structure has shown suitability to fairly wide range of programming tasks. However, relatively very few hypercube structures are implemented commercially. A nearest neighbor communication network is good for applications where the communications are restricted to neighboring PEs. However, there are several problems which require communications between PEs which are separated by a large distance. Massively parallel machines typically have a scheme to cover such communications patterns. The Connection Machine [4] has a hypercube ICN, the MasPar MP-1 [35] has an Omega network, DAP has row and column highways [23] and the DEC massively parallel processing chip has been designed with a router communication network [31]. These systems can solve the problem of irregular communications. However they integrate a fixed ICN. The problem is that different applications might have different demands for the architecture. In [15] author has demonstrated that the perfect shuffle interconnection pattern has a fundamental role in a parallel processor. It has been shown that for some examples including the Fast-Fourier Transform, polynomial evaluation, sorting, and matrix transposition, the shuffle interconnection scheme presents advantages over the near-neighbor (or cyclically symmetric) interconnection scheme that is used in the ILLIAC IV. So, the perfect shuffle interconnection scheme deserves to be considered for implementation in advanced parallel processors. Whether this interconnection pattern should be used instead of, or in addition to, other interconnection patterns depends very much on the size and intended application of the parallel processor. In [27], [14] a comparator study between four MINs for SIMD architecture (Feng's data manipulator, STARAN flip network, omega network, and indirect binary n-cube) is presented. According to this study, the networks may be ordered in terms of interconnection capabilities as follows: flip network, indirect binary n-cube, omega network, and ADM. The Omega network, for example, may have to use address transformations to perform all of the n-cube interconnections, due to the reversed order of its stages. It has been also shown that the omega, ncube, and ADM network have control structures which allow them to function in a multiple control environment. So, it has been proved that some types of MINs are good networks for SIMD systems.

This analysis shows that existing SIMD architectures are not capable of responding to the communication demands, especially of irregular type, of many data parallel algorithms. Hence, there is a need for an SIMD architecture that can perform neighbor as well as non neighbor communications integrating a NoC with good properties. An efficient and flexible irregular FPGA based communication network dedicated to massively parallel systems is described. A systemC implementation has already been proposed [33]. This work focuses rather on the RTL implementation and FPGA performance results of the mpNoC which can use different internal routers. We are interested by dynamic interconnection structures, in particular crossbar and Delta MIN. The validation of the mpNoC is performed through implementation and simulation.

3. Basic mppSoC architectural model

MppSoC [29], which stands for massively parallel processing System on Chip, is a novel SIMD architecture built within nowadays processors. It is composed of a grid of processors and memories connected by a X-Net neighbourhood network and a general purpose global router. MppSoC is an evolution of the famous massively parallel systems proposed at the end of the eighties. Each processor executes the very same instruction at each cycle in a synchronized manner, orchestrated by an unique control processor. The ACU is responsible for fetching and interpreting instructions.

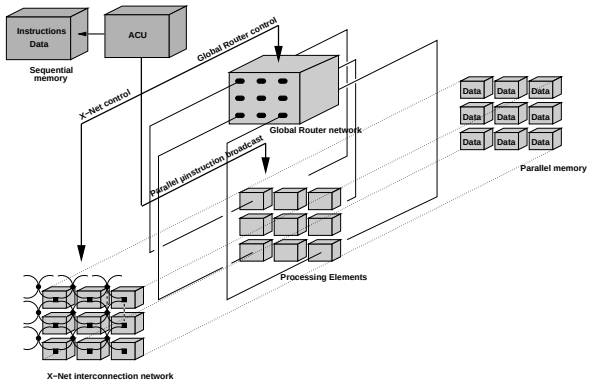

Figure 1: The mppSoC Architecture

Two kinds of instructions are considered: parallel ones and non-parallel (i.e. sequential) ones. The control processor transfers parallel arithmetic and data processing instructions to the processor array, and handles any control flow or serial computation that cannot be parallelized. Each PE in the 2-D grid is potentially connected to its 8 neighbors via the X-Net, a neighborhood network. However, this router cannot perform all the needed communications particularly irregular ones. So the idea is to integrate another communication network. A single shared bus between all PEs, for example, is not sufficient, since in an SIMD machine it is desirable to allow many processors to send data to other processors simultaneously. Ideally, one would like each processor sends data directly to every other processor, but this is highly impractical for large N, since each processor would require N-1 lines. So, one solution is the use of NoC. NoC-based communications will also become mandatory for many applications to enable parallel interconnections and communication throughputs [7]. In the mppSoC architecture, each PE is connected to an entry of mpNoC, a massively parallel Network on Chip that potentially connects each PE to one another, performing efficient irregular communications. The ACU synchronously controls the two networks of the system: the X-Net and mpNoC. Figure 1 illustrates the mppSoC global architecture. In this work, processors ACU and PEs are built from the processor IP miniMIPS [39] running at 50 Mhz. The ACU is a complete processor whereas the PE is a reduced processor derived from the same processor as the ACU [28].

MppSoC is programmed in a data-parallel language. A data-parallel language distinguishes sequential instructions and parallel instructions. A sequential instruction concerns sequential data of the ACU memory and is carried out by the ACU as in an usual sequential architecture. A parallel instruction is executed in a synchronous manner by all the PE of the system, each PE taking its operands from its local memory and storing the result in this same memory (or may be in its own local registers). Some specific instructions control the two networks, allowing transfer of values from one PE to another. Theses transfers are also executed in a synchronous manner: all PEs communicating at the same time with a PE designated by the instruction or one of its operand values. A given X-Net communication for example allows all PEs to communicate with a PE in a given direction at a given distance. Direction and distance are here the same for all the PEs.

As already mentioned, the design of mppSoC is inspired from the famous MasPar [35]. Nevertheless three major points distinguish mppSoC from the MasPar:

1. The mppSoC PEs are not any smaller 1- or 4-bit processors as it was by the time of the Connection Machine CM-1 and MasPar MP-1. MppSoC uses 32-bit processors.
2. The ACU and the PEs are designed from the same processor. Some minor additions are made to this processor to design the ACU, while its decode part is suppressed in the PE, performing a better on chip integration and reducing the power consumption.
3. The mppSoC global router not only connects the PEs to each others, but also allows connecting the PEs to ACU and to devices.

Another major difference between usual SIMD systems and our mppSoC is the integration of mpNoC, a multipurpose NoC component in the mppSoC. The mpNoC was designed as an IP that is able to synchronously connect a set of inputs to a set of outputs. It is based on a NoC router which transfers data from source to destination depending on the routing information. In this work, we detail the implementation of the mpNoC based on a crossbar and a Delta MIN. In the following section, we introduce the mpNoC and the routing mechanism. The used interconnection routers are also described.

4. MpNoC Design

MpNoC is the network component of the mppSoC that allows a parallel communication of each PE to a distinguished PE. One important property of the mpNoC is its ability to use different ICNs. An alternate network which allows all processors to communicate simultaneously is the crossbar switch. The difficulty here is that network costs grow with N^2; given current technology, this makes crossbar switches infeasible for large

systems. The full crossbar is used only with a small number of PEs. A more complex network is so needed for big instances of mppSoC. In our case we also test the integration of a Delta MIN in the mpNoC. MpNoC allows exchanging data between any couple of PE in parallel. It is considered as a global router of the mppSoC. Nevertheless, if any communication between PEs may be realized by a set of X-Net communications, as performances and flexibility are concerned, the usage of such a global router has an advantage over the X-Net usage for many algorithms. Consequently, including or not a global router in a given mppSoC design is a trade-off between the cost in term of silicon and the advantage in term of performance and flexibility, especially in the case of a design targeting a configurable hardware such as an FPGA. The nature of the targeted applications may be the decisive element in this design choice.

Communications IPs generally are tedious to integrate due to the number and the heterogeneity of connection they manage. The IP blocks are also connected into them through some fixed interface which will be highlighted. The following subsection deals with particularities of each IP block and its integration into mppSoC. We also demonstrate the flexibility of the mpNoC integration in the mppSoC system.

4.1. MpNoC Overview

The mpNoC is designed as an IP that can be configured to perform three functions in the mppSoC architecture. Firstly, the mpNoC is used as a global router connecting, in parallel, any PE with another one. Secondly, the mpNoC is able to connect the PEs to the mppSoC devices. Finally, the mpNoC is able to connect the ACU to any PE of the mppSoC [33]. It is usually impractical to implement all the interconnections that may be needed by the system to perform a large variety of computations, so the ability of a network to perform a variety of interconnections is important. The mpNoC includes an internal network which transfers data from sources to destinations. This network is the key point of mpNoC. In order to allow an efficient and a realistic IP integration of this network, its interface is generic enough to support a configurable size (4x4, 32x32 for example). While targeting an mpNoC integration into mppSoC, the number of mpNoC sources and destinations is equal to the number of PEs used in the mppSoC grid. PEs are not directly connected to the mpNoC but are connected to switches that allow to connect either the PEs, either the ACU, or some devices to the mpNoC. As shown in Figure 2, the mpNoC IP is connected to mppSoC and its input/output devices via controlled switches. These switches are also controlled via the

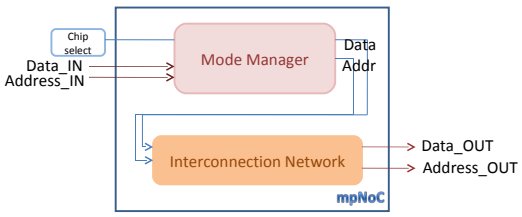

Figure 3: mpNoC Architecture

ACU based on the mode instruction. In fact, the mpNoC contains a Mode Manager which establishes the interconnections needed for one communication mode. There are three different bidirectional communication modes, as mentioned previously:

a) Mode PE - PE
b) Mode ACU - PE
c) Mode PE - I/O device

The designed mpNoC is parameterized in terms of number of PEs connected to the network and the chosen communication mode. The rest of this Section highlights the mpNoC implementation and covers crossbar and Delta multistage switching networks in detail.

4.2. MpNoC Implementation

MpNoC is implemented as a VHDL IP (Figure 3) composed of a Mode Manager, which is responsible of assuring the needed communication mode, and an ICN which is the router component that allows data transfer. This router could be of different types such as a crossbar or a MIN or other. However, integrating a given irregular NoC for a given application becomes tedious, error-prone, and time consuming due to the lack of a flexible and scalable interface. MpNoC is characterized by its flexible and scalable synchronous interface, in order to be integrated in different sized mppSoC configurations. Its interface is configured to transfer data from multiple senders (PEs/ACU/device) to multiple receivers (PEs/ACU/device). It is extensible to arbitrary number of ports. In fact, to support arbitrary numbers of PEs in an mpNoC interface, a simple approach is to statically map each PE request to one input port configured as a vector of length equal to the number of PEs. The mpNoC interface in one mppSoC configuration, with a VGA device for example, contains the following signals:

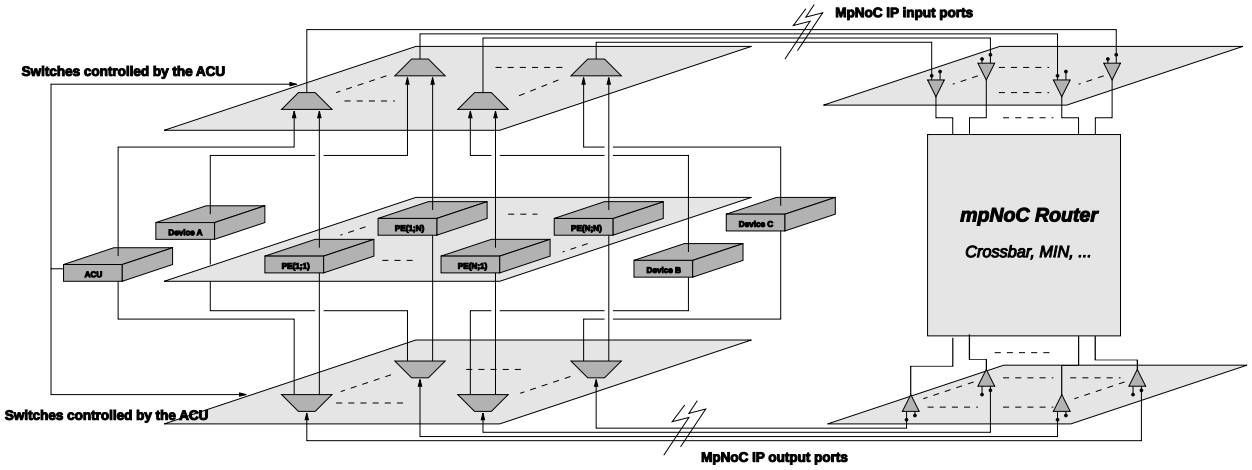

Figure 2: mpNoC integration into mppSoC

clock : clock signal
reset : reset signal
cs : activation bit
- - - - Processors - - - -
- - - - IN - - - -
datainPE : PE data (32bit data vector (length=number of PEs))
requestinPE : PE address (32bit address vector (length=number of PEs))
ram_wr_PE : PE read/write (1bit R/W vector (length=number of PEs))
datainACU : ACU data (32 bits)
requestinACU : ACU address (32 bits)
write_en : ACU read/write signal
- - - - OUT - - - -
dataoutPE : PE data Out (32bit data vector (length=number of PEs))
requestoutPE : PE address Out (32bit address vector (length=number of PEs))
dataoutACU : ACU data Out (32 bits)
requestoutACU : ACU address Out (32 bits)
- - - - Devices - - - -
dataoutVGA : VGA data Out (32 bits)
reqoutVGA : VGA address Out (32 bits)

The mpNoC always contains input and output ports dedicated to PEs and to ACU. The designer has to add only the necessary needed connections with the mppSoC devices. Such an interface can be easily incorporated into different mppSoC configurations. The internal router can service either a read or a write access at a time, as the address is either read or write address respectively, according to the read/write bit. If this bit is set to one, a write operation is performed; otherwise a read operation is achieved. The data transfer is arbitrated between all initiators in a round-robin fashion. To guarantee the synchronous functioning of the mpNoC, a controller is implemented. The Figure 4 illustrates how large the implemented mpNoC controller is in comparison to the different mpNoC components. We see that the size of the mpNoC controller is about 27% of the whole mpNoC design. The size of the Mode Manager is significantly lower than that of the router. It only represents 29% of the mpNoC design. So the major component

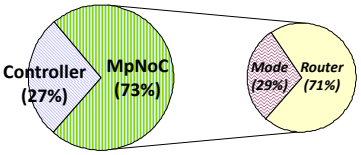

Figure 4: Size comparison of mpNoC components

that consumes more FPGA area is the internal router.

The following paragraphs detail more the mpNoC components.

4.2.1. Mode Manager

The Mode Manager is composed of switches responsible of establishing the needed connections according to the chosen communication mode. By default, the PE-PE mode is established. In fact, the mpNoC has as inputs the data coming from the PEs, the ACU and the devices. So the Mode Manager has to select the corresponding data depending on the communication mode set by the programmer. A chip select component may activate the mpNoC. The mode is selectable with a mode instruction.

4.2.2. Interconnection Network (mpNoC Router)

The ICN is the mpNoC router responsible of transferring data from sender to appropriate receiver. It may be of different types. It has scalable communication architecture in order to fit to different sized mppSoC architectures. In this work, two mpNoC implementations have been proposed: one is based on a full crossbar network, another on a Delta MIN. An effective implementation

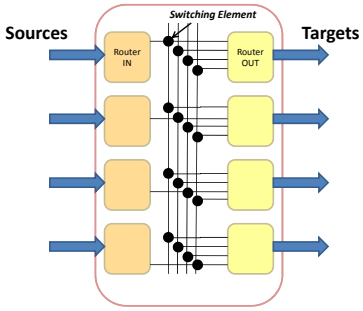

Figure 5: Example of 4x4 crossbar NoC

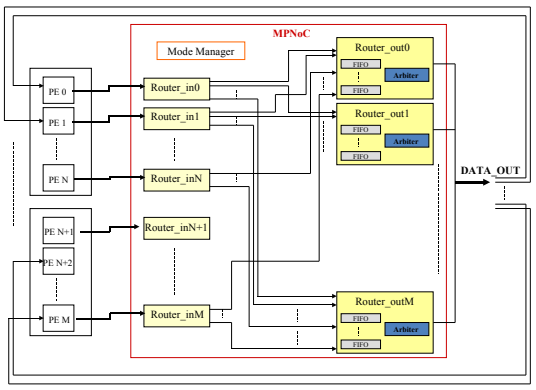

Figure 6: The crossbar architecture in the PE-PE mode

can be chosen depending, for instance, on the number of PEs. Our choice is based on the fact that crossbar networks are networks with good properties for systems with small number of PEs. While targeting a huge network size, it is necessary to deal between efficiency (expressed in average number of passes) and the silicon space. MIN is considered as a promising solution for applications which use parallel architectures integrating a large number of processors and memories. They meet the needs of intensive signal processing and they are scalable to connect a large number of modules. The mpNoC router has N input data ports and N input address ports, where N is the number of PEs in the system. If PEs are senders so all input ports are activated. If the ACU is the sender so the data of the ACU is transmitted via the first input port of the network by the Mode Manager. All the other input ports are disabled. The same manner is applied to the output ports. In the following sections, we discuss the implementation of each individual mpNoC internal network in detail. We functionally verified each individual block and synthesized each block and complete design using Quartus II synthesis tool and the simulator Modelsim Altera. We conclude with a comparison between the two implemented internal networks.

Crossbar based mpNoC

A full crossbar network allows simultaneously connecting any pair of nodes unoccupied. In general, it is used to connect a limited number of processors and memories. We look more closely into a packet-switched crossbar.

Implementation

The crossbar ICN is a multiple bus system, where all units are interlinked as shown in Figure 5. There is a separate path available to each target. The architecture is based on two main building blocks. Blocks on the left hand side are called Routers_IN, and blocks on the right hand side are called Routers_OUT. The Router_IN has the role of getting a request from its (left) input port and sending it to the right Router_OUT according to its address. Inversely, the Router_OUT detects if there is data to be sent to this port. When this is the case, a transfer occurs from the source to the target. The Figure 6 shows in detail the crossbar architecture connecting PEs to PEs (the case of the PE-PE mode). The network architecture consists of input ports, output ports, N routers, N arbiters (where N is the number of PEs in the system) and switch fabric. The switch fabric is the interconnection between inputs and outputs. The used ports communicate 32 bits wide data and address busses. Each Router_out stores the incoming data and address to which the data is destined in buffers. The buffers are maintained as First-In-First-Out (FIFO) queues. A common problem arises when several senders are ready to send their data to the same receiver. The round robin approach offers an elegant solution to fix this issue [36]. The Router_out contains an arbiter that assigns a priority token to one sender. As soon as a transfer occurs, the token goes to the next sender in a circular way. If one FIFO that has the priority is not ready, its nearest neighbor (in the round robin algorithm) that contains data will complete its transaction, and so on. In this way, a transfer can occur at each clock cycle for each Router_out that has at least one full FIFO. All the activities are synchronized to a global clock signal. Different buffer sizes are tested with the Router_out module. Table 1 shows the implementation results varying the buffer depth. It is clearly shown that using a buffer of less size consumes less area than when using a bigger size buffer. Since the mpNoC is dedicated to an SIMD architecture, it functions in a synchronous way. In this way, only one communication is executed at a time, then the next communication is executed after the comple-

Table 1: *Router_OUT* with different buffer sizes implementation results

buffer size	Logic Utilization		Total block
number of words	ALUTs	registers	memory bits
2	5	69	128
4	9	73	256
8	13	77	512
16	17	81	1024

tion of the first and so on. That's why, every buffer sufficiently needs two 32-bits wide words (data + address), even with a larger number of PEs. The final implemented crossbar contains FIFO buffers of size two. The following paragraph presents performance evaluation and analysis of the crossbar network.

Crossbar performances

In this paragraph, we evaluate the NoC performances. In fact, NoC evaluation metrics, such as area and average latency became essential aspects for optimizing the implementation of networks in a multi processor SoC design. The implementation results on the FPGA Altera Stratix 2S180, in terms of logic utilization, maximum throughput (TP) and latency are reported in Table 2. The TP is defined as the average number of data transfers delivered by the network per cycle per output port. The latency is the time spent to achieve a communication from the sender to the corresponding receiver. Minimum and maximum latency values are measured. We clearly see that when increasing the number of connected nodes, the crossbar network occupies more area on the target FPGA. One of other major difficulties with the crossbar is the rapid growth rate of the number of connections that must be made when new nodes are added. Crossbar has the complexity of N^2 connections, where N is the number of switches in this non blocking network. Test results also show that the max TP decreases as the number of connected nodes increases.

Delta MIN based mpNoC

A MIN can be defined as a network used to interconnect a group of N inputs to a group of M outputs using several stages of switches elements led by linking stages [8]. A MIN is defined by: its topology, switching strategy, routing algorithm, scheduling mechanism, and fault tolerance [37]. Among the proposed various ICNs, those most commonly used are the class of delta networks [16] which includes the omega network [12], indirect binary n-cube network [18], and the cube network [27]. Because of their low complexity, delta networks have always been considered as the alternatives to crossbar switches for interconnecting processors and

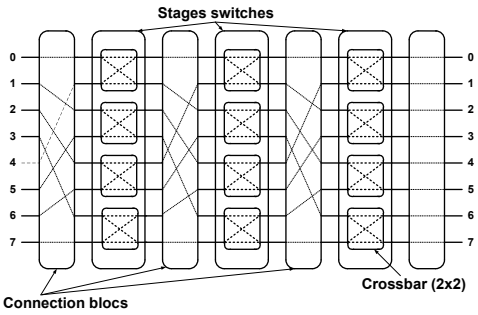

Figure 7: Delta MIN Architecture (Case of Omega network)

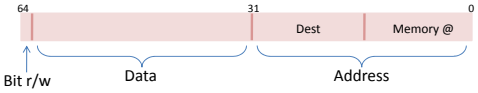

Figure 8: Delta MIN data packet

memory modules in multiprocessor systems [16]. In this work, we focus on Delta MIN which derived from Banyan networks characterized by one and only one path between each source and destination [9].

Implementation

General a^n x b^n delta network consists of a^n sources and b^n destinations, n number of stages and the i^{th} stage has a^{n-1} b^{n-1} crossbar modules or Switching Elements (SE) of size a x b (in our case crossbars of size 2x2). Figure 7 shows a three stages 2^3 x 2^3 switching Omega network. The nodes pairs are connected to each other with switches which are dynamically set by control logic associated with interconnected network. Delta network possesses full access property since it is possible to connect every node (N_i) to the other (N_j). In a MIN, a path between a source and a target is obtained by operating each corresponding switch of the stage i in straight mode if the i^{th} bit of the destination address is equal to 1, otherwise in exchange mode.

The basic building blocks of the Delta MIN are SEs, connected by links. The SE is composed of two FIFO and a scheduler who decides when data is sent from particular inputs to their desired outputs following a round robin scheduling algorithm. We can vary the topology of the network (omega, baseline, and butterfly) just by varying the topology of interconnection links between the crossbar stages. The implemented MIN is a packet data communication network. The package is composed of three parts, as shown in Figure 8. The 65-bits packet data is mainly composed of:

- *The head of the packet (1 bit)*: contains the bit

Table 2: Crossbar performance results

Number	Logic Utilization		Total block memory	Latency	Max TP
PEs	ALUTs	registers	bits	(cycles)	(32bits/cycle)
4PEs	281	483	512	2-5	1.232
8PEs	548	806	1024	2-9	1.131
16PEs	957	1469	2048	2-17	1.045
32PEs	1905	2818	4096	2-33	0.923
64PEs	4191	5917	8192	2-65	0.833
128PEs	10184	13609	16384	2-129	0.782

Table 3: Delta MIN performance results

Number	Logic Utilization		Latency	Max TP
PEs	ALUTs	registers	(cycles)	(32bits/cycle)
4	402	1404	6-9	1.260
8	1101	3885	9-16	1.193
16	1456	5401	12-27	1.085
32	2099	7746	15-46	0.886
64	2905	9022	18-81	0.812
128	3834	10397	21-148	0.742

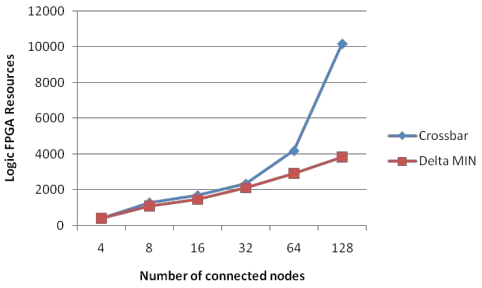

Figure 9: Logic utilization of the crossbar and Delta MIN networks

read/write to determine the nature of the memory access read/write.

- *The data (32 bits)*

- *The tail of the packet (32 bits)*: composed of the destination and the memory address.

Delta MIN performances

Simulation results give statistical values for FPGA area, latency and the maximum throughputs of the blocking network which are reported in Table 3. As the crossbar, the Delta MIN requires more FPGA resources when increasing the number of connected nodes. We clearly see from the above tabulated results that the data transfer rate decreases as the number of nodes increases. One major difference is the network latency which is higher in the MIN compared to the crossbar.

Crossbar/Delta MIN Comparison

The implementation of the two internal networks shows that the mpNoC is a flexible and an efficient network. The choice of the utilized network should be based on the application requirements. The most important metrics helping the decision process are interconnection area and TP. Comparing between the two different mpNoC ICNs, as shown in Figure 9, we notice that the area of a full crossbar network is increasingly important than MIN while increasing the size of the two networks (more than 64 connected nodes). The area of interconnection depends on the topology (topology weakly connected will have an area smaller than

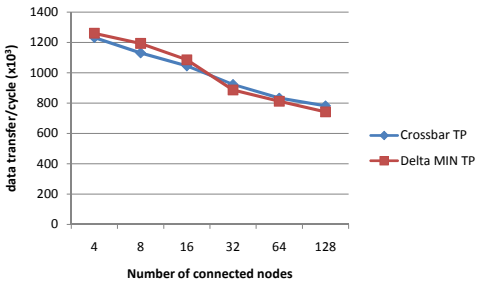

Figure 10: Maximum TP of the two internal mpNoC networks

topology completely connected), located services (more the established mechanisms are complex and numerous, more resources area is important) and the size of buffers included in the routing resources. As a result, Delta MINs are performing in terms of area to connect a large number of nodes. The area of MIN is proportional to Nlog_2N, compared to N^2 for full crossbar networks. In Figure 10, the obtained maximum TPs of a crossbar and a Delta MIN networks with different number of nodes are compared. Results demonstrate that the TP of the delta MIN and the crossbar are comparable. The delta MIN has a TP higher than the crossbar with smaller number of PEs (less than 32 PEs), and vice-versa. According to these results, it seems that the delta MIN is better suited for systems with small trans-

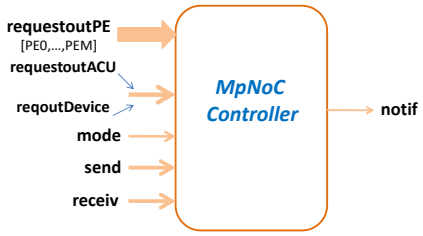

Figure 11: MpNoC Controller Interface

missions. When transmissions become longer, crossbar has a higher TP which is mostly constituted by the fact, that there are more interconnection paths available in parallel. Whereas, delta networks suffer from internal blocking which severely degrades their TP performance.

We deduce that the crossbar network offers multiple simultaneous communications with the least amount of contention, but at a very high hardware complexity. The crossbar is a low latency, non-blocking ICN for large data transfers. However, it becomes expensive for large values of N. In comparison, multistage switching networks may offer a better cost (area and power)/performance (delay and TP) compromise for large complex systems.

4.2.3. MpNoC Controller

To assure synchronization, which is the key characteristic of an SIMD system, an mpNoC controller IP must be integrated in the architecture when using the mpNoC. It has a functioning dependent on the communication mode. This IP verifies if data transferred by a sender is received by the corresponding receiver. The verification is carried when there are multiple receivers or multiple senders to assure all data transmissions. Otherwise, the single receiver sends directly an acknowledgement to the ACU. The mpNoC controller (Figure 11) has as inputs: the addresses signals transferred via the mpNoC (*requestoutPE* and *requestoutACU* or *reqoutDevice*), the communication mode *mode*, *send* and *receiv* signals coming from senders and receivers respectively; and a notification signal *notif* as output port. *send* and *receiv* signals are configured as bit arrays of length equal to the number of PEs as it is the maximum number of connections of the mpNoC. If the sender is the ACU or a device, it has to set the first bit of the *send* signal to '1', otherwise it is set to '0'. The same manner is followed if we only have one receiver which can be the ACU or a device. If the PE is the sender it has to set the bit which has the same number as its identity to '1'. The same manner is applied by the PE receiver. For example in the case of 4 PEs, *send* and *receiv* signals are configured as 4-bits arrays (*array (nb_slave-1 downto 0) of std_logic; where nb_slave is equal in this case to 4*). If the ACU sends data to the PE1 then the ACU has a *send* signal configured as "0001" and the PE1 has a *receiv* signal configured as "0010" when receiving its data. The mpNoC controller continuously verifies if the communication is achieved form senders to appropriate receivers. It also verifies if all needed communications are successful. Based on the mode communication, it compares between *send* and *receiv* signals taking into account the transmitted addresses via the mpNoC. When the communication is successful, the mpNoC controller generates a notification signal to the ACU in order to continue executing the remaining instructions.

In order to allow sending and receiving data through networks, we use different communication instructions that will be described in the following subsection.

4.3. Communication instructions

We use the MIPS assembly language for development of mppSoC parallel programs. From an mppSoC assembly code, the mppSoC compiler generates a binary that can be used by the FPGA implementation. This mppSoC compiler is a modification of the GNU MIPS assembler. MpNoC is managed through communication instructions based mainly on the processor load (LW) and store (SW) instructions. To set the communication mode we employ a mode instruction which consists in writing the mode value in a defined address. It is based on the SW processor instruction: *SW cst, @ModeManager*, where:

- *@ModeManager* = "0x9003"
- *cst* is a constant which can have five different values corresponding to five different interconnection modes respectively.

Below, the definition of mode constant values in the VHDL configuration file is showed.

```
–MPNoC Modes
constant Mode0 : positive := 0; – PE – > PE
constant Mode1 : positive := 1; – ACU – > PE
constant Mode2 : positive := 2; – PE – > ACU
constant Mode3 : positive := 3; – PE – > Device
constant Mode4 : positive := 4; – Device – > PE
```

From an implementation point of view the communication mode constant is a three bits value. After setting the required mpNoC interconnection, data transfers will occur through communication instructions which are SEND and RECEIVE instructions.

SEND instruction: serves to send data from a sender to a corresponding receiver, relying on the SW memory instruction: *SW data, address*. The 32-bits address can be partitioned in different fields depending on the established communication mode. It contains in case of:

1. PE-PE Mode: the identity of the PE receiver (32-SL_add_width bits) and the PE dest memory address (SL_add_width bits);
2. PE-ACU Mode: the ACU memory address (MS_add_width bits);
3. ACU-PE Mode: the identity of the PE receiver (32-SL_add_width bits) and the PE dest memory address (SL_add_width bits);
4. PE-Device Mode: the device address (32 bits);
5. Device-PE Mode: the identity of the PE receiver (32-SL_add_width bits) and the PE dest memory address (SL_add_width bits).

MS_add_width and SL_add_width are the parametric memory sizes of the ACU and the PE memories respectively. When needed, zeros are inserted into the emptied bits to obtain the 32-bits address. The communication modes ACU-Device and Device-ACU are direct point to point communications.

RECEIVE instruction: serves to obtain the received data, relying on the LW memory instruction: *LW data, address*. It analogously takes the same address field as SEND instruction.

It is clear from the above address coding that the number of PEs in the system also depends on the memory size. However, increasing the number of PEs leads to decreasing the PE memory size which is the advantage of the SIMD architecture. For example when each PE has a memory of 256 K-bytes we can integrate up to 65536 PEs, which is a sufficient number.

Furthermore, we notice that the execution time of a communication instruction can vary depending on the communication mode. If multi-senders want to send their data to a same receiver (PE-ACU mode for example), the data is sequentially transferred since the receiver can accept one data at a time. So the total execution time of this communication instruction is the execution time considering one to one communication multiplied by the number of senders (N) since we have to repeat it N times (we find the same process in the traditional SIMD systems like Maspar [35]).

The Xnet neighborhood network is programmed as the mpNoC. It is managed by SEND and RECEIVE instructions. Their address field takes three operands: the distance, the direction and the memory address. The distance defines the number of paths needed to achieve the communication between the PE sender and the other receiver on the same row or column or diagonal. There are eight direction 3-bits constant values that the programmer can specify to denote each of the following directions: North (000), East(010), South(001), West(011), North East(100), North West(101), South East(110) and South West(111).

Table 4: SW MIPS Format

31 – 26	25 – 21	20 – 16	15 – 0
101011	base	rt	offset

Table 5: SEND Format

31 – 26	25 – 21	20 – 16	15 – 0
101011	rd	rt	offset

SEND and RECEIVE instructions rely on LW and SW MIPS instructions with some added modifications. In fact the SW instruction has the format shown in Table 4. In this format, the address is composed of 21 bits (offset+base) and the value to be stored is contained in the register rt. In comparison, the SEND instruction replaces the base field, as shown in table 5, by the number of the register (rd) that can contain the address to be transmitted. In this case, the address field is the sum of the register rd value and the offset, and is 32 bits wide.

5. MppSoC Applications

The objective of this section is to test the use of different communication networks: from a neighborhood network to an irregular network with different interconnection routers. We want to test the reliability and the effectiveness of the NoC and compare the performances of the various parallel configurations with distinct communication networks. Two mpNoC networks were tested: full crossbar and Delta MIN with three topologies (Omega, Baseline and Butterfly). These latter differ in the number of different N-to-N interconnection patterns they can achieve. Three parallel algorithms were implemented: image rotation, 2D convolution and FIR Filter. There are two main considerations in the mapping of parallel algorithms onto mppSoC. First is the number of available PEs and the second is the selection of the best configuration to map the algorithm. The purpose of this Section is to evaluate performance of three parallel algorithms when mapped onto mppSoC system. Implementation results and performance evaluation of each algorithm are presented. The analysis provides the flexibility to vary several parameters, and therefore, it is easier to study the effects of alternative approaches. The results identify the communi-

Figure 12: Extract from image rotation assembly codes

cation network suitable for an algorithm. As already mentioned, the MIPS instruction set was extended to program the mppSoC [28]. The execution of mppSoC binary needs first to integrate the binary program in the instruction memory and, after the synthesis, is deployed with the bitstream to the FPGA board. We target Altera Stratix 2S180 FPGA which includes 143 520 ALUTs for hardware logic [38].

5.1. Image Rotation Algorithms

Image rotation algorithms seem to be simple and good examples to use the mpNoC. In fact, in this situation, communications are very irregular: PEs need to communicate using several different directions and lengths. Obviously, it is possible to realize that using the X-Net network, but this needs several communication steps (as much as the number of PEs). We realize 17161-pixel image rotations on different sized mppSoC designs. The resulting images of Figure 12 were provided by an execution of binary programs on mppSoC mapped to an FPGA. In this work, we are specifically interested by image mirroring and special image rotation by 90, 180 and 270 degrees. It is quite simple to do these rotations by remapping the pixel locations: choosing the source pixel that corresponds to each destination pixel and setting the destination pixel to that value. In these algorithms we are considered that the number of PEs is equal to N where N=n^2. The PEs are arranged in a 2D (nxn) grid. As an example for an image rotation (of size (MxM)) by 90 degrees, each PE performs the rotation on its submatrice of size (M/nxM/n). Then all PEs send their data in order to the VGA device to reconstruct the final rotated image. The mpNoC is used to connect PEs, ACU with PEs and to also connect PEs with a VGA device allowing displaying image on the screen. So, the mpNoC is necessary in this application since it assures parallel I/O data transfers. The following code shows the reconstruction of the rotated image by the VGA device:

```
(ST1) For i=0 to n-1 do
(ST2) For j=0 to n-1 do
(ST3) Read data from PE(i)
(ST4) i:= i+n
(ST5) end For j
(ST6) end For i
```

Table 6: Clock frequency for mppSoC on Stratix 2S180 FPGA

Number of PEs	Max. Freq. (MHz) (mppSoC with crossbar)	Max. Freq. (MHz) (mppSoC with Omega)
4	53.39	54.3
8	53.04	53.46
16	51.27	51.77
32	48.63	41.68
64	41.48	40.70

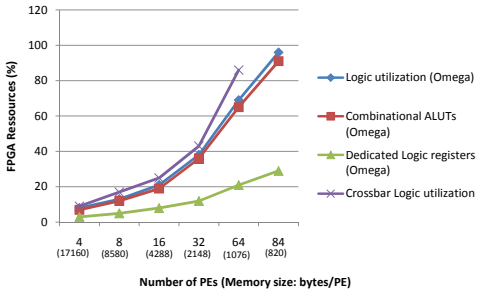

Figure 13: Synthesis results of Delta MIN/crossbar based mppSoC designs

When varying the number of PEs, we also vary the data memory size of each PE. Two internal mpNoC ICNs are tested: a crossbar and a Delta MIN with three topologies (Omega, Butterfly and Baseline). The table 6 presents the maximum clock frequency for the different mppSoC designs varying the number of processors and the used mpNoC interconnection networks: crossbar and omega networks. Clock frequency drops due to the used network. Although the frequency decreases, all designs approximately run at the first frequency which is 50 MHz. The mppSoC with the omega network runs at a frequency higher that when using the crossbar for a number of PEs lower than 16. This conforms the results obtained in table 2 and 3. Performance results in terms of computation cycles, logic area and energy consuming are also analysed. Figure 13 demonstrates that 84 PEs could be implemented if using Delta MIN configured in the Omega topology, on the StratixII. However, when using the crossbar based mpNoC we are limited by the huge size of the crossbar network and 64 is the maximum number of PEs that we could integrate on the StratixII FPGA. It is also the same number that we could integrate when using both neighbouring and mpNoC networks in the same system. The table 7 gives the resources occupation for the ACU and the PE. It is shown that one of the advantages of the SIMD parallel system is a saving in the amount of logic. About 30% of the logic on a typical processor chip is devoted to

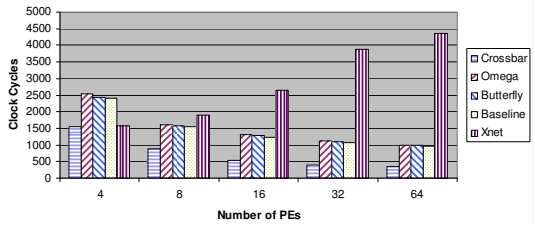

Figure 14: Execution time of image rotation algorithm for different sized mppSoC designs

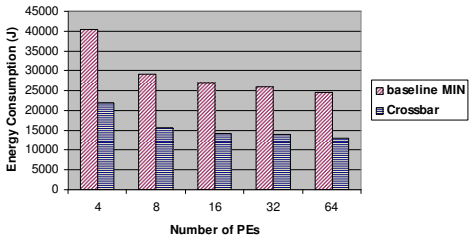

Figure 15: Energy Consumption of different sized mppSoC designs

Table 7: Processors FPGA logic utilization

Processors	ALUTs	registers	FPGA occupation
ACU	2851	1936	2%
PE	1031	321	<1%

control.

As illustrated by Figure 14, the speedup increases when increasing the number of PEs. SIMD systems can provide a high throughput, as long as the processing algorithm exhibits a high degree of parallelism at the instruction level. However, this is not the case when using the Xnet network since the PE spend more time to do communication than computation. Since communications are irregular, we need many cycles to achieve all the communications between PEs. It is so shown that the Xnet network is not efficient for the image rotation. This is due to the fact that the communications are not regular enough to be managed by the Xnet network. We demonstrate that the irregular communications are very tedious to realize using the neighboring network. We deduce that the crossbar based mppSoC design performs a speed up higher than a Delta MIN based mppSoC design. It has also been shown that the baseline topology is the most appropriate topology if the designer wants to implement a Delta MIN based mpNoC with this application. The three different MIN topologies differ in terms of the connection links between the crossbar based stages.

In order to estimate the embedded system efficiency, we measure the amount of energy E =TxP required to compute the algorithm, where P(W) is the power dissipation. P values are measured using the PowerPlay power analysis tool of Quartus II. It can be observed from the Figure 15 that, as we increase the number of PEs, the energy consumption decreases. We clearly see that the crossbar based mppSoC consumes less energy than a baseline MIN based mppSoC. This is due to the fact that the crossbar achieves less execution time. According to these different performance results, we deduce that the choice of a crossbar internal network is better, in our case, than a Delta MIN, depending on the application requirements. It should be also noted that the complexity of the crossbar network pays off in the forms of reduction in the time complexity as well as energy consuming. However, if the designer wants to integrate more than 64 PEs in the mppSoC system, he should choose a Delta MIN based mpNoC.

5.2. 2D Convolution Algorithm

Two-dimensional (2D) convolution is a basic operation in image processing and requires intensive computation. The SIMD model is considered suitable for this kind of application. In fact, the image convolution involves local image transformations resulting in thousands of potentially parallel operations. We focus on the 2D convolution computed in the discrete wavelet transform (DWT) [20]. In this work, we perform convolution on a 256x256 32-bit pixels image. Multiple pixels are mapped onto a separate PE. For an image NxN, each processor has M=N^2/P pixels in its local memory, where P (assume P=pxp) is the total number of PEs. In general, pixel(i,j), 0≤i≤N-1, 0≤j≤N-1 is mapped to PE((i mod p),(j mod p)). Therefore this mapping preserves the adjacency of any two pixels. In an initialization phase, the ACU sends to every PE the corresponding pixels to be stored in its local data memory. The convolution operation is performed as follows: for each pixel(i,j) a 2x2 sliding template, called convolution kernel, is convolved with the 2x2 window centered on pixel(i,j). That is, each value into the pixels window is multiplied by the corresponding signed weight into the convolution kernel. Then, the 2x2 products obtained in this way are added to produce the output pixel value. Each PE has the kernel coefficients stored in its local data memory. The algorithm performs the convolution by each processor distributing its pixel values to the neighborhood in a pipelined manner. In this case, the Xnet neighborhood network is used. But only North, South, East and West connections are required. At any

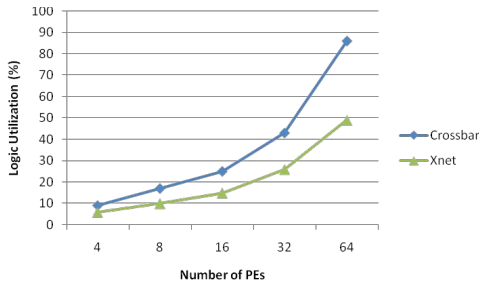

Figure 16: FPGA Resources of Crossbar/Xnet based mppSoC designs

step all PEs have the same neighbor connection. We also test the use of a crossbar based mpNoC and compare it with the neighborhood network.

In this case, the relationship between the input pixels B_s(x,y) (each PE performs a convolution on its bloc image denoted by B_s) , the convolution kernel weights h(i,j) and the convolved pixels r_s(x,y) is given by:

$$r_s(x,y) = B_s * h$$
$$= \sum_{i=1}^{n}\sum_{j=1}^{n} B_s(x-i+1, y-j+1) \times h(i,j) \quad (1)$$

where $s \in [1, nb\_PEs]$; n=2

At any time, every PE computes a partial sum for its convolution. The implemented algorithm is similar to that explained in [19], [1]. We address the scaling of mppSoC to match the computational complexity of a convolution application. The impact of scaling is quantified in terms of FPGA allocation, execution time and energy consuming. As shown in Figure 16, the crossbar network is considerably larger, especially in the implementation with 64 PEs. With 4 PEs the crossbar is about 1,5 times as big as the Xnet, but with 64 PEs this relationship is increased to twice. It is therefore likely that this difference in size will increase further when implementing larger networks.

The neighborhood network is more efficient for applications which needs inter processor communications since it is designed for that purpose. As illustrated from Figure 17, we can see that the regular network version is the fastest one. For 64 PEs for example, the mppSoC with Xnet takes 14 ms whereas with a crossbar it takes 25 ms which is approximately two times higher. In addition, the Xnet network has a latency of one cycle, due to local communications, making it a more powerful and efficient communication network to perform neighboring communications compared to mpNoC. Figure 18 demonstrates that the Xnet consumes less energy than

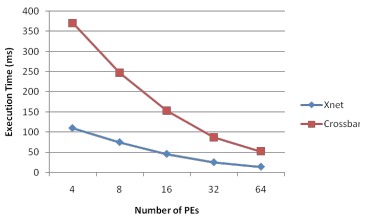

Figure 17: Execution Time of 2D convolution algorithm

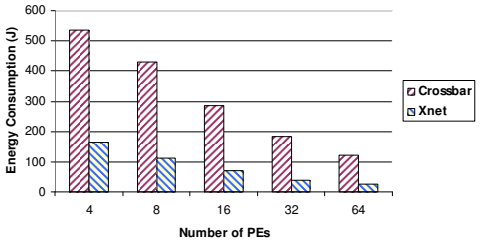

Figure 18: Energy Consuming for Crossbar/Xnet mppSoC designs

the crossbar. Consequently, using a neighborhood network for a 2D convolution is more suitable than integrating the mpNoC.

5.3. FIR Filter

FIR (Finite Impulse Response) filtering is one of the most popular DSP algorithms. It is well suited to be executed on SIMD systems. FIR filters are easy to design. On the other hand, they require increased number of multiplications and additions, and, what is also important, number of memory reads. A FIR filter is implemented with the following equation:

$$Y(n) = \sum_{k=0}^{M-1} b_k X(n-k) \quad (2)$$

where X is the input signal and b_k are filter coefficients. M is the filter order or number of taps. An M-order FIR filter requires M multiplications and M additions for every output signal sample. It also requires 2M memory read operations, M of them is for input signal, the rest for them is for filter coefficients. In this work an adapted version of the difference equation called the Direct Form Structure (Figure 19), is implemented. To run the FIR-filter, we tested two mppSoC designs based on two different communication networks: the neighborhood network Xnet and the crossbar. Application results describe the performance and speedup of the implemented FIR-filter. Shown in Figure 20 are the clock cycles needed when running the FIR filter application.

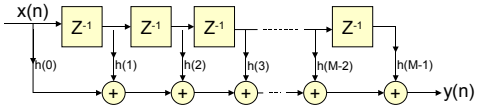

Figure 19: Direct Form of FIR Filter

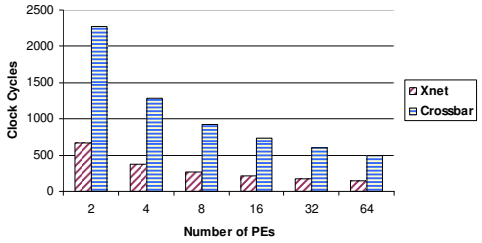

Figure 20: Execution Time of a FIR algorithm for different sized mppSoC designs

The results are based on a 64-tap FIR and an impulse response with a length of 128. When PEs are added to the system, a larger part of the output signal can be calculated on at the same time. On the other hand, communication instructions are decreased and this results in a maximum speed up of about five. As expected, the mppSoC architecture based on the neighbourhood inter-processor network is the most effective for the FIR application. These results show that, based on flexible communication networks in the mppSoC system, the programmer would use the best interconnect suited to his application and its requirements.

It has been proved from these experiments that availability of flexible communication is critical to achieve high performance.

6. Network architecture comparison

Many NoC architectures proposals have been investigated. The challenge consists in offering the best connectivity and throughput with the simplest and cheapest architecture, particularly for parallel architectures as SIMD ones. Unfortunately, very few of these proposals provided any kind of implementation or performance data that could be used for relevant comparison against this work. In addition few NoC implementations have been proposed for SIMD parallel systems. As a result it is difficult to perform direct comparison with other SIMD dedicated network approaches. However, other NoC implementations for multiprocessing/parallel systems could be representative to be compared with our mpNoC.

Table 9: MCNoC and mpNoC performance results

NoC Version	Occupation		Throughput
	FFs	LUTs	
MCNoC	3951	4731	280 MB/s
mpNoC	806	548	305,115 MB/s

Our proposal is the massively parallel network on chip, mpNoC, which is flexible and can implement different interconnection networks. This flexibility is a key characteristic of mpNoC that distinguishes it from the other proposed NoC. It also supports different communication modes: one to many, many to one and many to many communications offering parallel data transfers. The mpNoC is scalable and can cope with a large number of cores. Comparing it with some other NoC implementations, we find that our mpNoC is efficient and provides powerful performances.

For example, in [22] a multiprocessor architecture for the massively parallel GCA model is presented. It contains an omega network. The table 8 compares between this network and our omega MIN based mpNoC. We see that our network presents better results when increasing the number of PEs since it consumes less ALUTs in the FPGA (reduction of the hardware cost by over 3x with 32 cores). This makes it efficient for massively parallel on chip architectures. In [30] a parallel routing mechanism for a MIN on the circuit-switching mode is presented. The authors take into account only one-to-one and one-to-many permutations. However, many-to-one and many-to-many permutations are not considered. Our approach differs from the previous one. In fact, our target architecture is an SIMD massively parallel architecture where multicast has an important role, and one-to-one permutation almost never takes place. The proposed mpNoC can be also configured to support different communication modes by programming. Through experiments, we show that a configurable parallel router that can change its communication mode in accordance with the applications communication need can potentially increase the performance of the NoC and the final system. In [25] a multi-cluster NoC architecture for parallel processing is proposed. It shows better performance than conventional NoCs. When comparing the area occupation and the network throughput for 8 cores with our crossbar based mpNoC, we find that the mpNoC achieves better results, as demonstrated in table 9 . Accroding to table 9, the mpNoC presents a high throughput than the MCNoC (1.089x). In [21] a programmable NoC is proposed. It has a flexible architecture dedicated to be used in FPGA-based systems. It

Table 8: Synthesis Results

Network	Comm. accesses	ALUTs			
		4PEs	8PEs	16PEs	32PEs
Omega	Only read	279	818	2344	6092
mpNoC	Read and write	402	1101	1456	2099

Table 10: PNoC and mpNoC router characteristics

Network	Flexibility	Parameter	Switching
PNoC	Network topology	Data-path width & connected nodes	circuit
mpNoC	Interconnection network	connected nodes	packet

Table 11: PNoC and mpNoC performance results

Number of nodes	Network	Area	Speed (MHz)
4	PNoC	366 slices	138
4	mpNoC (crossbar)	445 Logic Elements	123
8	PNoC	1305 slices	126
8	mpNoC (crossbar)	802 Logic Elements	113

also shows better performances compared with a shared bus implementation. The Table 10 compares PNoC and our mpNoC in terms of router characteristics. The significant difference is the network parameters. The actual mpNoC version works with 32bits data width. We note that ongoing work aims to parametrize the data width of the mpNoC, however it is outside the scope of this paper. The table 11 compares PNoC and crossbar based mpNoC, working with 32bits data width, in terms of area and speed for 4 and 8 connected nodes. We see that the mpNoC achieves better performances in term of area since it reduces the hardware cost when increasing the number of connected nodes. However it decreases slightly the speed. In general, it presents good area/time performances.

We deduce from previous comparisons that the proposed mpNoC is well suited for parallel architectures achieving better performances. It is considered as a lightweight network which requires few FPGA resources making it suitable for both small and large FPGA-based systems.

7. Conclusion

Having an efficient communication network in modern multiprocessor systems on-chip is certainly one of the biggest challenges for designers. This is particularly true for SIMD architectures. In this paper, we introduced the mppSoC system which is an SIMD massively parallel processing System on Chip. Among its important features, we emphasize on its communication networks. The mppSoC platform uses X-Net network for inter-processor communications. This network is very efficient for neighboring communications. However, it is time consuming when dealing with point-to-point communications. For that purpose, mppSoC also integrates an efficient irregular communication network IP called mpNoC, a massively parallel Network on Chip that can be used alone or with X-Net.

The mppSoC platform is entirely described at RTL level and implemented on FPGA. It can be configured to use different sizes and network topologies. This configurability makes it possible to tailor the architecture for a specific application and thereby increasing its effectiveness. MpNoC can be configured in different communications modes: to communicate between processors and also to perform I/O data transfer. This work shows the gains that could be achieved with such strategy. MpNoC uses a set of library internal networks that have varying cost and performance metrics. Two networks were tested: a crossbar and a Delta MIN with three different topologies (omega, baseline and butterfly). The mppSoC designed networks are scalable and parametric in order to satisfy different data parallel application requirements. The FPGA implementation, with various configurations, has been validated on three significant applications. According to the performances of each configuration, the designer can choose the most

appropriate one for the tested application. We have demonstrated that it is vital to have a flexible interconnection scheme which can be applied to the system design. Compared to other implemented NoC, the mpNoC presents good performances making it suitable to FPGA based parallel systems.

The implementation and evaluation of significant and complete applications on mppSoC system are ongoing. Future works deal with the choice of the processor IP. The MIPS processor has been chosen for our first implementation because of its open source availability; however, it is not optimized for a particular FPGA. Our aim is to test other processor IPs more optimized in order to enhance the implementation effectiveness. Finally, on the integration side, if large configuration may not be integrated on a single chip, we are considering multichip implementations. Connecting together on a board, those chips will be able to act like a unique SIMD machine executing a single program. The definition of a chip interface and especially the splitting of the networks on the different chips has to be studied with a special attention on the scalability of the architecture.

References

[1] A. N. Choudhary and J. H. Patel, Parallel architectures and parallel algorithms for integrated vision systems (University of Illinois at Urbana-Champaign, 1989).

[2] B. Parhami, Introduction to Parallel Processing: Algorithms and Architectures (Kluwer Academic Publishers, 1999).

[3] R. Michael Hord, The Illiac IV: The First Supercomputer (Computer Science Press, 1982).

[4] W. D. Hillis, The Connection Machine (The MIT Press, Cambridge, 1989).

[5] M-H. Lee, H. Singh, G. Lu, N. Bagherzadeh, F. J. Kurdahi, E. M. C. Filho and V. C. Alves, Design and Implementation of the MorphoSys Reconfigurable Computing Processor. J. VLSI Signal Processing Systems. 24 (2000) 147-164.

[6] M. Kumar and J.R. Jump, Performance Enhancement in Buffered Delta Networks Using Crossbar Switches and Multiple Links. J. Parallel and Distributed Comput. (1) (1984) 81-103.

[7] M. Rupp, D. Milojevic and G. Gogniat, Design and Architectures for Signal and Image Processing. EURASIP J. on Embedded Systems. (2008).

[8] T. H. Szymanski and V. C. Hamacher, On the universality of multipath multistage interconnection networks, J. Parallel and Distributed Comput. 7 (1989) 541-569.

[9] C. P. Kruskal and M. Snir, A unified theory of interconnection network, Theoret. Comput. Sci. 48 (1986) 75-94.

[10] C. P. Kruskal and M. Snir, The Performance of Multistage Interconnection Networks for Multiprocessors. IEEE Trans. Comput. 32 (1983) 1091-1098.

[11] C. P. Kruskal, M. Snir, and A. Weiss, The distribution of waiting times in clocked multistage interconnection networks. IEEE Trans. Comput. 37 (1988) 1337-1352.

[12] D. H. Lawrie, Access and Alignment of Data in an Array Processor. IEEE Trans. Comput. 24 (1975) 1145-1155.

[13] D.M. Dias and J. R. Jump, Analysis and Simulation of Buffered Delta Networks. IEEE Trans. Comput. 30 (1981) 273-282.

[14] H. J. Siegel, Interconnection Networks for SIMD Machines, Computer. 12 (1979) 57-65.

[15] H. S. Stone, Parallel Processing with the Perfect Shuffle. IEEE Trans. Comput. 20 (1971) 153-161.

[16] J. H. Patel, Performance of processor-memory interconnections for multiprocessors. IEEE Trans. Comput. 30 (1981) 771-780.

[17] L. Benini and G. DeMicheli, Networks on Chips: A New SoC Paradigm. IEEE. Computer. 35 (2002) 70-78.

[18] M. C. Pease, The Indirect Binary n-Cube Microprocessor Array. IEEE Trans. Comput. 26 (1977) 458-473.

[19] S. Ranka and S. Sanhi, Convolution on Mesh Connected Multicomputers. IEEE Trans. on Pattern Analysis and Machine Intelligence. 12 (1990) 315-318.

[20] A. Al Muhit, Md S. Islam and M. Othman, VLSI Implementation of Discrete Wavelet Transform (DWT) for Image Compression, in: Proc. International Conference on Autonomous Robots and Agents, ICARA'04 (New Zealand, 2004).

[21] C. Hilton and B. Nelson, PNoC: a flexible circuit-switched NoC for FPGA-based systems, in: Proc. IEEE Computers and Digital Techniques (2006) 181-188.

[22] C. Schack, W. Heenes and R. Hoffmann, A Multiprocessor Architecture with an Omega Network for the Massively Parallel Model GCA, in: Proc. 9^{th} International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (2009) 98-107.

[23] D. Parkinson, Experience in Using Highly Parallel Processing Using DAP, in: Proc. Massively Parallel Scientific Computation (NASA Conference Publication, 1986) 205-208.

[24] F. Schurz and D. Fey, A Programmable Parallel Processor Architecture in FPGAs for Image Processing Sensors, in: Proc. Integrated Design and Process Technology, IDPT'07 (2007).

[25] H. C. Freitas and P. O. A. Navaux, Evaluating On-Chip Interconnection Architectures for Parallel Processing, in: Proc. 11^{th} International Conference on Computational Science and Engineering - Workshops (2008) 188-193.

[26] H. Du, M. Sanchez-Elez, N. Tabrizi, N. Bagherzadeh, M. L. Anido and M. Fernandez, Interactive ray tracing on reconfigurable SIMD MorphoSys, in: Proc. Design, Automation and Test in Europe Conference, DATE'03 (2003).

[27] H. J. Siegel and S. D. Smith, Study of multistage SIMD interconnection networks, in: Proc. 5^{th} annual Symp. Computer Architecture (1978) 223-229.

[28] M. Baklouti, Ph. Marquet, M. Abid and JL. Dekeyser, A design and an implementation of a parallel based SIMD architecture for SoC on FPGA, in: Proc. Design and Architectures for Signal and Image Processing DASIP (2008).

[29] Ph. Marquet, S. Duquennoy, S. Le Beux, S. Meftali and JL. Dekeyser, Massively parallel processing on a chip, in: Proc. 4^{th} International Conf. Computing Frontiers (2007) 277-286.

[30] R. Ferreira, M. Laure, A. C. Beck, T. Lo, M. Rutzig and L. Carro, A Low Cost and Adaptable Routing Network for Reconfigurable Systems, in: Proc. IEEE International Symposium on Parallel & Distributed Processing IPDPS (2009) 1-8.

[31] R. Grondalski, A VLSI Chip Set for a Massively Parallel Architecture, in: Proc. International Solid State Circuits Conference, ISSCC'87 (1987).

[32] S. D. Smith and H. J. Siegel, An Emulator Network for SIMD Machine Interconnection Networks, in: Proc. 6^{th} annual symposium on Computer architecture (1979) 232-241.

[33] S. Duquennoy, S. Le Beux , Ph. Marquet , S. Meftali and JL. Dekeyser, MpNoC Design: Modeling and Simulation, in: Proc. 15^{th} IP Based SoC Design Conference, IP/SOC (2006).

[34] S. E. Eklund, A Massively Parallel Architecture for Linear Machine Code Genetic Programming, in: Proc. 4^{th} International Conference on Evolvable Systems: From Biology to Hardware

(2001) 216-224.
[35] T. Blank, The MasPar MP-1 Architecture, in: Proc. IEEE Compcon Spring90 (IEEE Society Press, San Francisco, CA, 1990) 20-24.
[36] X. Gao, Z. Zhang and X. Long, Round Robin Arbiters for Virtual Channel Router, in: Proc. Multiconference on Computational Engineering in Systems Applications, IMACS (2006) 1610-1614.
[37] Y. Aydi, S. Meftali, M. Abid and JL. Dekeyser, Dynamicity Analysis of Delta MINs for MPSOC Architectures, in: Proc. International Conference on Sciences ans Techniques of Automatic control and computer engineering, STA'07 (2007).
[38] Altera Corporation, Stratix II Device Handbook. (2004).
[39] OpenCores, miniMIPS overview. <http://www.opencores.org/project,minimips>.